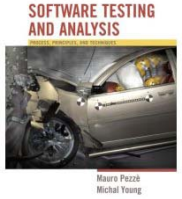
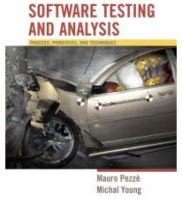
System, Acceptance, and Regression Testing

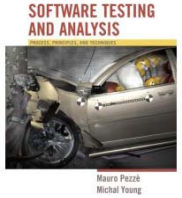


Learning objectives

- Distinguish system and acceptance testing
  - How and why they differ from each other and from unit and integration testing
- Understand basic approaches for quantitative assessment (reliability, performance, …)
- Understand interplay of validation and verification for usability and accessibility
  - How to continuously monitor usability from early design to delivery
- Understand basic regression testing approaches
  - Preventing accidental changes

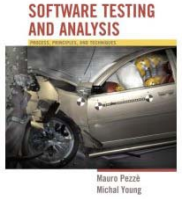


	System	Acceptance	Regression
Test for ...	Correctness, completion	Usefulness, satisfaction	Accidental changes
Test by ...	Development test group	Test group with users	Development test group
	Verification	*Validation*	Verification



22.2

System testing

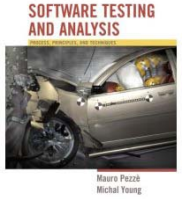


System Testing

- Key characteristics:
  - Comprehensive (the whole system, the whole spec)
  - Based on specification of observable behavior

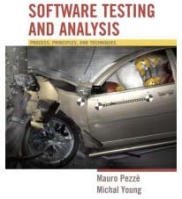
    Verification against a requirements specification, not validation, and not opinions
  - Independent of design and implementation

Independence: Avoid repeating software design errors in system test design



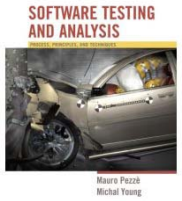
Independent V&V

- *One strategy for maximizing independence:* System (and acceptance) test performed by a different organization
  - Organizationally isolated from developers (no pressure to say “ok”)
  - Sometimes outsourced to another company or agency
    - Especially for critical systems
    - Outsourcing for independent judgment, not to save money
    - May be *additional* system test, not replacing internal V&V
  - Not all outsourced testing is IV&V
    - Not *independent* if controlled by development organization



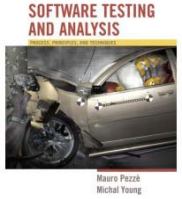
Independence without changing staff

- If the development organization controls system testing …
  - Perfect independence may be unattainable, but we can reduce undue influence
- Develop system test cases early
  - As part of requirements specification, before major design decisions have been made
    - Agile "test first" and conventional "V model" are both examples of designing system test cases before designing the implementation
    - An opportunity for "design for test": Structure system for critical system testing early in project



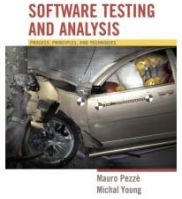
Incremental System Testing

- System tests are often used to measure progress
  - System test suite covers all features and scenarios of use
  - As project progresses, the system passes more and more system tests
- Assumes a "threaded" incremental build plan: Features exposed at top level as they are developed



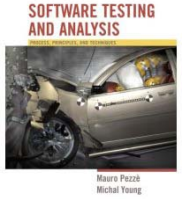
Global Properties

- Some system properties are inherently global
  - Performance, latency, reliability, …
  - Early and incremental testing is still necessary, but provide only estimates
- A major focus of system testing
  - The only opportunity to verify global properties against actual system specifications
  - Especially to find unanticipated effects, e.g., an unexpected performance bottleneck



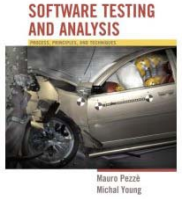
Context-Dependent Properties

- Beyond system-global: Some properties depend on the system context and use
  - Example: Performance properties depend on environment and configuration
  - Example: Privacy depends both on system and how it is used
    - Medical records system must protect against unauthorized use, and authorization must be provided only as needed
  - Example: Security depends on threat profiles
    - And threats change!
- Testing is just one part of the approach



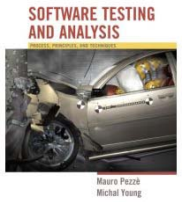
Establishing an Operational Envelope

- When a property (e.g., performance or real-time response) is parameterized by use …
  - requests per second, size of database, …
- Extensive stress testing is required
  - varying parameters within the envelope, near the bounds, and beyond
- Goal: A well-understood model of how the property varies with the parameter
  - How sensitive is the property to the parameter?
  - Where is the “edge of the envelope”?
  - What can we expect when the envelope is exceeded?



Stress Testing

- Often requires extensive simulation of the execution environment
  - With systematic variation: What happens when we push the parameters? What if the number of users or requests is 10 times more, or 1000 times more?
- Often requires more resources (human and machine) than typical test cases
  - Separate from regular feature tests
  - Run less often, with more manual control
  - Diagnose deviations from expectation
    - Which may include difficult debugging of latent faults!



Capacity Testing

- **When:** systems that are intended to cope with high volumes of data should have their limits tested and we should consider how they fail when capacity is exceeded
- **What/How:** usually we will construct a harness that is capable of generating a very large volume of simulated data that will test the capacity of the system or use existing records
- **Why:** we are concerned to ensure that the system is fit for purpose say ensuring that a medical records system can cope with records for all people in the UK (for example)
- **Strengths:** provides some confidence the system is capable of handling high capacity
- **Weaknesses:** simulated data can be unrepresentative; can be difficult to create representative tests; can take a long time to run

Security Testing

- **When:** most systems that are open to the outside world and have a function that should not be disrupted require some kind of security test. Usually we are concerned to thwart malicious users.
- **What/How:** there are a range of approaches. One is to use league tables of bugs/errors to check and review the code (e.g. SANS top twenty-five security-related programming errors). We might also form a team that attempts to break/break into the system.
- **Why:** some systems are essential and need to keep running, e.g. the telephone system, some systems need to be secure to maintain reputation.
- **Strengths:** this is the best approach we have most of the effort should go into design and the use of known secure components.
- **Weaknesses:** we only cover known ways in using checklists and we do not take account of novelty using a team to try to break does introduce this.

Performance Testing

- **When:** many systems are required to meet performance targets laid down in a service level agreement (e.g. does your ISP give you 2Mb/s download?).
- **What/How:** there are two approaches - modelling/simulation, and direct test in a simulated environment (or in the real environment).
- **Why:** often a company charges for a particular level of service - this may be disputed if the company fails to deliver. E.g. the VISA payments system guarantees 5s authorisation time delivers faster and has low variance. Customers would be unhappy with less.
- **Strengths:** can provide good evidence of the performance of the system, modelling can identify bottlenecks and problems.
- **Weaknesses:** issues with how representative tests are.

Compliance Testing

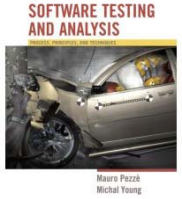
- **When:** we are selling into a regulated market and to sell we need to show compliance. E.g. if we have a C compiler we should be able to show it correctly compiles ANSI C.
- **What/How:** often there will be standardised test sets that constitute good coverage of the behaviour of the system (e.g. a set of C programs, and the results of running them).
- **Why:** we can identify the problem areas and create tests to check that set of conditions.
- **Strengths:** regulation shares the cost of tests across many organisations so we can develop a very capable test set.
- **Weaknesses:** there is a tendency for software producers to orient towards the compliance test set and do much worse on things outside the compliance test set.

Documentation Testing

- **When:** most systems that have documentation should have it tested and should be tested against the real system. Some systems embed test cases in the documentation and using the doc tests is an essential part of a new release.
- **What/How:** test set is maintained that verifies the doc set matches the system behaviour. Could also just get someone to do the tutorial and point out the errors.
- **Why:** the user gets really confused if the system does not conform to the documentation.
- **Strengths:** ensures consistency.
- **Weaknesses:** not particularly good on checking consistency of narrative rather than examples.

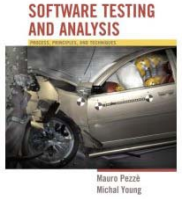
22.3

Acceptance testing



Estimating Dependability

- Measuring quality, not searching for faults
  - Fundamentally different goal than systematic testing
- Quantitative dependability goals are statistical
  - Reliability
  - Availability
  - Mean time to failure
  - …
- Requires valid statistical samples from *operational profile*
  - Fundamentally different from systematic testing

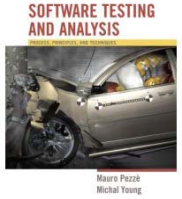


Definitions

- **Reliability**: Survival Probability
    - When function is critical during the mission time.
- **Availability**: The fraction of time a system meets its specification.
    - Good when continuous service is important but it can be delayed or denied
- **Failsafe**: System fails to a known safe state
- **Dependability**: Generalisation - System does the right thing at right time

Statistical Sampling

- We need a valid *operational profile* (model)
  - Sometimes from an older version of the system
  - Sometimes from operational environment (e.g., for an embedded controller)
  - *Sensitivity testing* reveals which parameters are most important, and which can be rough guesses
- And a clear, precise definition of what is being measured
  - Failure rate? Per session, per hour, per operation?
- And many, many random samples
  - Especially for high reliability measures



System Reliability

- The reliability, $R_F(t)$ of a system is the probability that no fault of the class F occurs (i.e. system survives) during time t.

$$R_F(t) = P(t_{init} \leq t < t_f \forall f \in F)$$

where t_{init} is time of introduction of the system to service,
t_f is time of occurrence of the first failure f drawn from F.

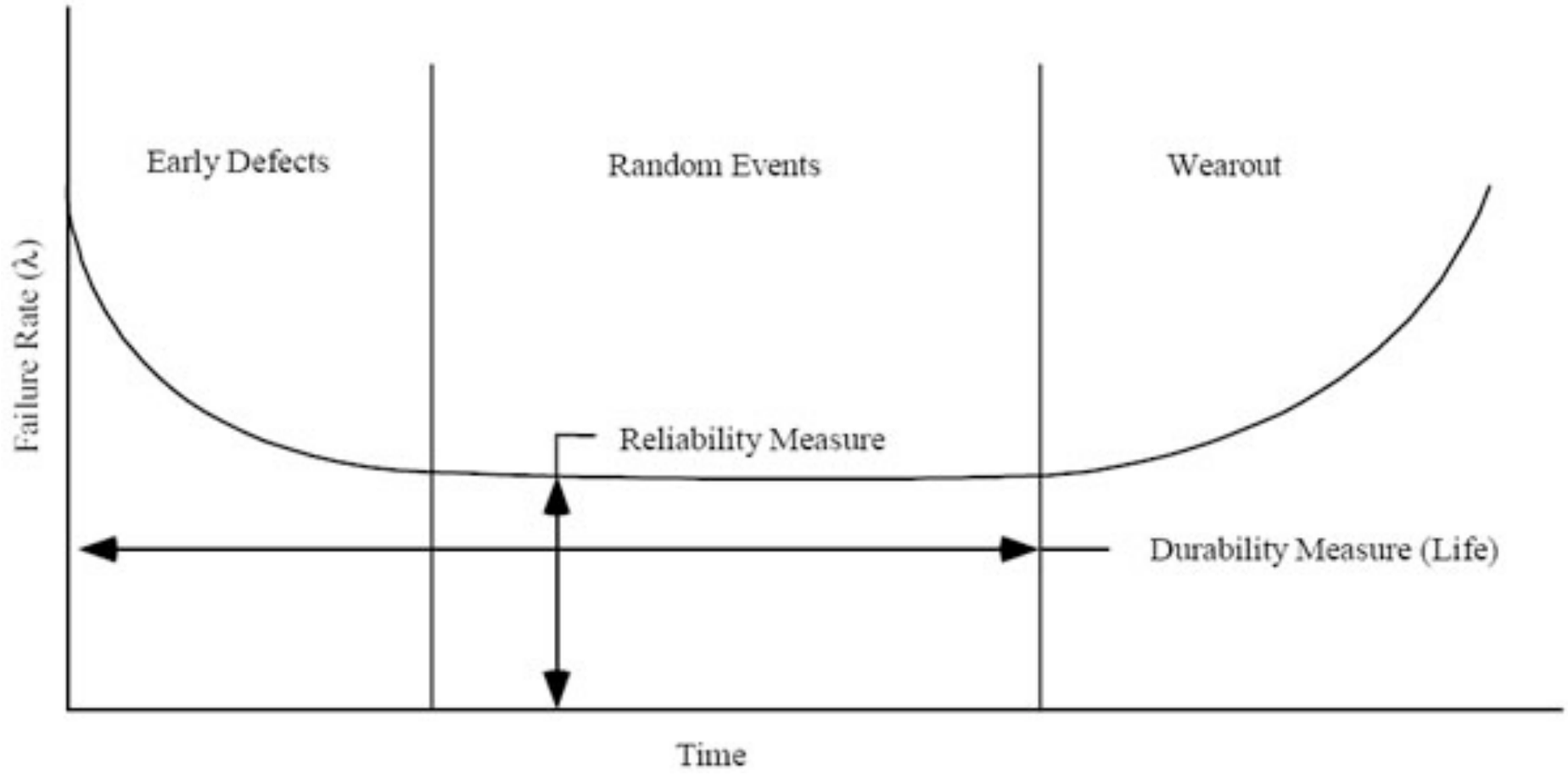
- Failure Probability, $Q_F(t)$ is complementary to $R_F(t)$

$$R_F(t) + Q_F(t) = 1$$

- We can take off the F subscript from $R_F(t)$ and $Q_F(t)$

- When the lifetime of a system is exponentially distributed, the reliability of the system is: $R(t) = e^{-\lambda t}$ where the parameter λ is called the failure rate

Component Reliability Model



Burn In Useful Life Wear Out

During useful life, components exhibit a constant failure rate λ. Reliability of a device can be modelled using an exponential distribution $R(t) = e^{-\lambda t}$

Component Failure Rate

- Failure rates often expressed in failures / million operating hours

Automotive Embedded System Component	Failure Rate λ
Military Microprocessor	0.022
Typical Automotive Microprocessor	0.12
Electric Motor Lead/Acid battery	16.9
Oil Pump	37.3
Automotive Wiring Harness (luxury)	775

MTTF: Mean Time To Failure

- **MTTF**: Mean Time to Failure or Expected Life

- **MTTF**: Mean Time To (first) Failure is defined as the expected value of t_f

$$MTTF = E(t_f) = \int_0^\infty R(t)dt = \frac{1}{\lambda}$$

  where λ is the failure rate.

- **MTTF** of a system is the expected time of the first failure in a sample of identical initially perfect systems.

- **MTTR**: Mean Time To Repair is defined as the expected time for repair.

- **MTBF**: Mean Time Between Failure

Serial System Reliability

- Serially Connected Components
- $R_k(t)$ is the reliability of a single component k: $R_k(t) = e^{-\lambda_k t}$
- Assuming the failure rates of components are statistically independent.
- The overall system reliability $R_{ser}(t)$

$$R_{ser}(t) = R_1(t) \times R_2(t) \times R_3(t) \times \ldots \times R_n(t)$$

$$R_{ser}(t) = \prod_{i=1}^{n} R_i(t)$$

- No redundancy: Overall system reliability depends on the proper working of each component

$$R_{ser}(t) = e^{-t(\sum_{i=1}^{n} \lambda_i)}$$

- Serial failure rate

$$\lambda_{ser} = \sum_{i=1}^{n} \lambda_i$$

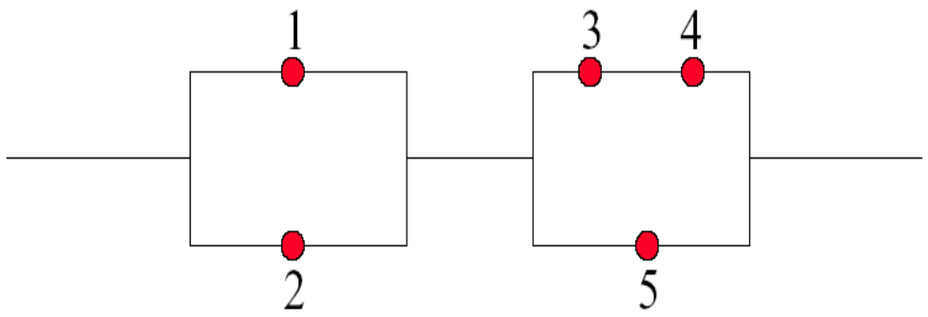
System Reliability

- Building a reliable serial system is extraordinarily difficult and expensive.

- For example: if one is to build a serial system with 100 components each of which had a reliability of 0.999, the overall system reliability would be

$$0.999^{100} = 0.905$$

- Reliability of System of Components



- Minimal Path Set:
  Minimal set of components whose functioning ensures the functioning of the system: {1,3,4} {2,3,4} {1,5} {2,5}

Parallel System Reliability

- Parallel Connected Components
- $Q_k(t)$ is $1 - R_k(t)$: $Q_k(t) = 1 - e^{-\lambda_k t}$
- Assuming the failure rates of components are statistically independent.

$$Q_{par}(t) = \prod_{i=1}^{n} Q_i(t)$$

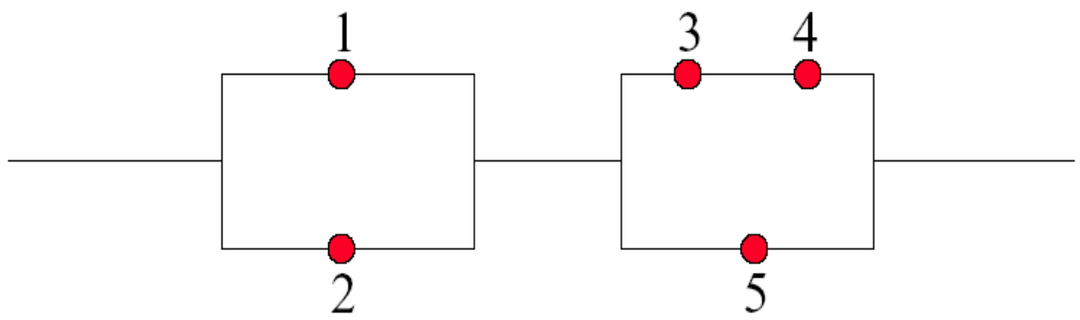
- Overall system reliability: $R_{par}(t) = 1 - \prod_{i=1}^{n}(1 - R_i(t))$

Example

- Consider 4 identical modules are connected in parallel
- System will operate correctly provided at least one module is operational. If the reliability of each module is 0.95.
- The overall system reliability is $1 - (1 - 0.95)^4 = 0.99999375$

Parallel-Serial Reliability

- Parallel and Serial Connected Components



- Total reliability is the reliability of the first half, in serial with the second half.

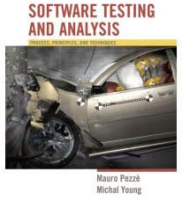
- Given R1=0.9, R2=0.9, R3=0.99, R4=0.99, R5=0.87

- $R_t = (1 - (1 - 0.9)(1 - 0.9))(1 - (1 - 0.87)(1 - (0.99 \times 0.99))) = 0.987$

Is Statistical Testing Worthwhile?

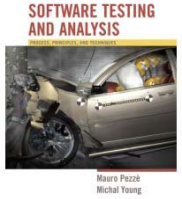
- Necessary for …
  - Critical systems (safety critical, infrastructure, …)

- But difficult or impossible when …
  - Operational profile is unavailable or just a guess
    - Often for new functionality involving human interaction
      - But we may factor critical functions from overall use to obtain a good model of only the critical properties
  - Reliability requirement is very high
    - Required sample size (number of test cases) might require years of test execution
    - Ultra-reliability can seldom be demonstrated by testing



Process-based Measures

- Less rigorous than statistical testing
  - Based on similarity with prior projects
- System testing process
  - Expected history of bugs found and resolved
- Alpha, beta testing
  - Alpha testing: Real users, controlled environment
  - Beta testing: Real users, real (uncontrolled) environment
  - May statistically sample users rather than uses
  - Expected history of bug reports



Usability Testing

- **When:** where the system has a significant user interface and it is important to avoid user error — e.g. this could be a critical application e.g. cockpit design in an aircraft or a consumer product that we want to be an enjoyable system to use or we might be considering efficiency (e.g. call-centre software).
- **What/How:** we could construct a simulator in the case of embedded systems or we could just have many users try the system in a controlled environment. We need to structure the test with clear objectives (e.g. to reduce decision time,...) and have good means of collecting and analysing data.
- **Why:** there may be safety issues, we may want to produce something more useable than competitors' products...
- **Strengths:** in well-defined contexts this can provide very good feedback – often underpinned by some theory e.g. estimates of cognitive load.
- **Weaknesses:** some usability requirements are hard to express and to test, it is possible to test extensively and then not know what to do with the data.

Reliability Testing

- **When:** we may want to guarantee some system will only fail very infrequently (e.g. nuclear power control software we might claim no more than one failure in 10,000 hours of operation). This is particularly important in telecommunications.
- **What/How:** we need to create a representative test set and gather enough information to support a statistical claim (system structured modelling supports demonstrating how overall failure rate relates to component failure rate).
- **Why:** we often need to make guarantees about reliability in order to satisfy a regulator or we might know that the market leader has a certain reliability that the market expects.
- **Strengths:** if the test data is representative this can make accurate predictions.
- **Weaknesses:** we need a lot of data for high-reliability systems, it is easy to be optimistic.

Availability/Reparability Testing

- **When:** we are interested in avoiding long down times we are interested in how often failure occurs and how long it takes to get going again. Usually this is in the context of a service supplier and this is a Key Performance Indicator.
- **What/How:** similar to reliability testing – but here we might seed errors or cause component failures and see how long they take to fix or how soon the system can return once a component is repaired.
- **Why:** in providing a critical service we may not want long interruptions (e.g. 999 service).
- **Strengths:** similar to reliability.
- **Weaknesses:** similar to reliability – in the field it may be much faster to fix common problems because of learning.

Summary

- There are a very wide range of potential tests that should be applied to a system.
- Not all systems require all tests.
- Managing the test sets and when they should be applied is a very complex task.
- The quality of test sets is critical to the quality of a running implementation.